Reducing Temporal Interference in Private Clouds through Real-Time Containers

Scuola Superiore Sant'Anna, Pisa, Italy
T. Cucinotta, L. Abeni, M. Marinoni, A. Balsini

Ericsson AB, Stockholm, Sweden
C. Vitucci

***Abstract*—Providing innovative resource-efficient solutions able to mitigate temporal interference among cloud services, concurrently sharing the same underlying platform, is crucial to deploy highly time-sensitive applications at the edge of the network where resources are strongly restrained, and timing constraints are stringent. A notable example is provided by the allocation of virtualized network functions in the radio access network of modern mobile networks, such as 5G.**

This paper describes a kernel mechanism that can be applied to the design of an architecture providing fine-grain control of the temporal interferences among concurrent real-time services while avoiding overheads related to machine virtualization. On top of them, a model is proposed to meet the required end-to-end application performance through tuning of parameters in the underlying novel architecture. We show that theoretical latency/load curves match closely with experimental data gathered from a real implementation carried out using both a networking microbenchmark and a real IMS application.

***Keywords*-cloud computing, edge computing, real-time scheduling, temporal isolation, performance modeling, network function virtualization**

I. Introduction

Information and communication technologies have undergone a steady evolution in recent years, with a massive shift toward distributed computing. Cloud computing, coupled with the widespread diffusion of broadband Internet connections, induced a paradigm transformation towards more and more services provisioned through Cloud Computing infrastructures, in an on-demand and elastic fashion, to meet consumers' needs rapidly evolving towards high reliability, high availability, and high performance.

Cloud infrastructure providers have a growing concern regarding the efficient management of hosted services, which led to a broad and variegated amount of innovations in the last few years. These comprise hardware features implemented by manufacturers as hardware-assisted virtualization mechanisms to reduce overheads due to machine virtualization, software solutions focused on tuning hosted operating systems internals and software stacks, stirring them towards para-virtualization, and other mixed solutions based on library operating systems or unikernels [1], [2]. Among all these heterogeneous innovations, a notable trend interests distributed software architectures based on containers [3], micro-services [4] or even server-less deployments [5].

The exploitation of such improvements concerning isolation and predictability in distributed (private) cloud infrastructures is attracting the attention of players that found the limitations of current cloud computing solutions too taxing for their application constraints. A significant use case is provided by network operators who are interested in creating infrastructures allowing dynamic and distributed provisioning of network functions following the Network Function Virtualization (NFV) paradigm, which aims to deploy network functions as software components instead of employing traditional physical devices. In these scenarios, the locality of placement plays a crucial role, since stringent latency constraints characterize the deployed Virtualized Network Functions (VNFs), thus demanding efficient underlying software solutions. Promising solutions to meet these goals rely on operating system (OS)-level [6] virtualization techniques to improve performance. These solutions can be realized for example using LXC[1], LXD[2] or Docker[3], able to present almost no overhead with respect to bare-metal solutions, while maintaining the key features of virtual machines, like isolation.

A. Problem presentation

The allocation of containers or Virtual Machines (VMs) in infrastructures like those previously illustrated must deal with the issue of temporal interference when their number surpasses the amount of available physical CPUs (pCPUs) in the host, i.e., multiple virtual CPUs (vCPUs) must share the same physical CPU, or in the presence of bottlenecks affecting other resources (e.g., disks or networks) due to data-intensive workloads.

The current approach for isolating computations is based on avoiding oversubscription, applying a 1-to-1 assignment from vCPUs to pCPUs. However, this forces the allocation granularity to be equal to the single pCPU, leading to potential waste of computational resources, particularly in the presence of small or highly variable workloads assigned to the single container or VM.

Hence, the infrastructure provider is bound to decide its allocation approach between: a) performing 1-to-1 pinning of vCPUs to pCPUs, and undergo the consequent resources waste; b) implementing pCPU sharing among vCPUs (oversubscription), with substantial performance instability due to the arising interferences. Both ways are not practicable in the nodes at the edge of the network, where resources are highly constrained, energy efficiency can be a crucial issue, and respecting severe timing constraints is essential for applications. For example, in the illustrated NFV scenario, oversubscription impacts on the quality of service (QoS) during

[1]More information at: https://linuxcontainers.org/lxc/introduction/.
[2]More information at: https://linuxcontainers.org/lxd/introduction/.
[3]More information at: https://docs.docker.com/.

traffic peak hours, while 1-to-1 allocation unacceptably raises energy consumption, that represents a predominant portion of the energy budget of access networks.

B. Paper contributions and structure

This paper presents key features needed to reach the level of fine-grain allocation and the corresponding modeling that are mandatory to fruitfully apply cloud computing infrastructures to time and resource constrained applications at the edge of the network. In particular, new hierarchical scheduling features of the SCHED_DEADLINE real-time scheduler are exploited at the OS kernel level, in order to apply the resource reservation paradigm to isolate whole containers. These kernel extensions let the OS guarantee the assignment of precise pCPU fragments to each application with an exact per-container time granularity, making the performance of the hosted services more stable and predictable. This is fundamental for building a performance model of the distributed service/application. As an example, a performance model based on queuing theory is built, considering both networking and processing reservations, for a synthetic distributed application with packet inter-arrival times, packet sizes and per-packet processing times all being i.i.d. with exponential distributions. The obtained model allows for providing an accurate estimate of the application performance, as shown via extensive experimentation on synthetic and IMS application scenarios running on Linux.

The paper is structured as follows. Section II depicts an overview of related works in the current research literature. Section III introduces the proposed approach, accompanied by the proposed modeling effort for a synthetic client-server application detailed in Section III-B. Section IV validates the proposed approach and the described application model through extensive experimentations performed with a patched Linux kernel that extends the mainline SCHED_DEADLINE CPU scheduler [7] with hierarchical scheduling. The solution is also applied to a node hosting a simple NFV function to show its applicability in a simple but real scenario. Finally, Section V draws conclusions and outlines possible future work on the topic.

II. Related Work

The cloud approach has been able to widen its application range due to its positive features making it fit to bring advantages to a heterogeneous collection of scenarios. However, constraints like latency and power consumption are more critical at the edge of the network than in a typical cloud scenario, and must wisely be considered. Hence, novel mechanisms able to enforce a more fine-grained control of resource usage at OS and hypervisor levels are critical. Also, they must be coupled with an equally detailed model of the platform and resource availability, so that precise application timing constraints can be met.

Various works exist on controlling performance of distributed cloud services via elasticity and auto-scaling mechanisms [8], [9], intelligent placement strategies [10], [11], possibly including network-awareness [12] and SDN-based approaches [13]. To reduce interferences of co-located services, solutions based on real-time scheduling applied to hypervisors have been proposed, e.g., for the KVM [14] and Xen [15] hypervisors.

Some authors suggested to accelerate cloud infrastructures exploiting heterogeneous hardware platforms, including GP-GPUs [16] and FPGAs [17]. However, this is orthogonal to the problem of limiting temporal interferences among co-located services, discussed in this paper. A wide range of performance-related features is available in operating systems and hypervisors nowadays, needing to be extended and exposed in order to make them usable by higher layers actors, such as cloud orchestration [18].

In the world of telecommunications, network operators are extensively using cloud principles for developing NFV solutions, and are interested in extending the paradigm up to the edge of the network [19]. For example, the Virtualized Radio Access Network (VRAN) seeks to move computational functionalities of the networking stack from the radio head(s) to dedicated servers at the edge of the infrastructure [20], [21], despite the stringent timing constraints sometimes in place, like the one of $4ms$ imposed on the acknowledgment of packets by the Hybrid ARQ (HARQ) [22] protocol. This solution is in line with the constant demand for reducing energy consumption in the deployment of VRAN, adopting a wise allocation of processing elements.

In [15], authors propose to enhance the Xen hypervisor with new features to support real-time processing, including the application of hierarchical real-time scheduling theory [23] and allocation of precise time slices of the physical CPU to each VM running on the platform. The same authors also realized an extension [24] to OpenStack exploiting the experimental features introduced at the hypervisor level. These Xen extensions have also been employed for creating a real-time NFV solution [25]. However, these solutions introduce genuine virtualization with all the associated overheads that are missing in a container-based solution. Real-time hierarchical scheduling strategies have been introduced for the Linux kernel, e.g., in [14], [26] that can be employed with containers if extended to manage `cgroups` [27].

These enabling mechanisms can provide an adequate guarantee to the time constraints of hosted services, but they become much more effective when coupled with proper modeling and analysis methods, as proposed in this paper.

III. Proposed approach

This paper considers the kind of systems discussed in [28], and also shown in Figure 1, composed of multiple clients submitting requests to a group of servers. Requests can be served by one single server selected by a load balancer, might need to traverse a chain of servers, or might be subject to more involved distributed processing topologies.

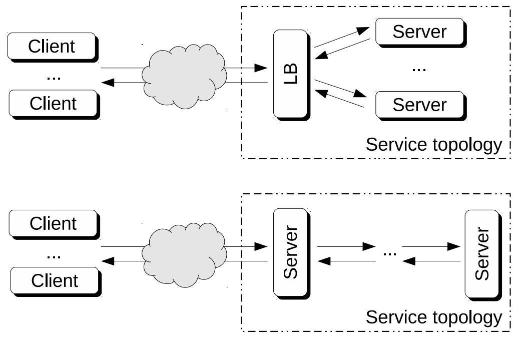


Figure 1. Reference service topologies.

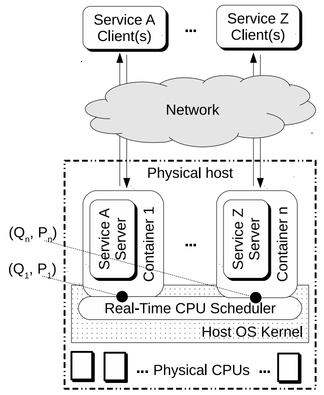


Figure 2. Node architecture for the proposed approach: n service containers are deployed over a host with m physical CPUs (with $m < n$). Each container is configured with custom scheduling parameters (Q_i, P_i).

Traditional cloud computing systems try to control the QoS experienced in these topologies through elastic scaling (horizontal, or, less frequently, vertical) and/or load balancing techniques implemented in the cloud orchestration layer. Unfortunately, this approach can be used only with services spanning across multiple instances and is not effective when multiple services are co-located on the same physical nodes and share some resources (for example, the CPU). In this case, in fact, the execution of each service can be preempted in a non-controllable way by other services running on the same CPU, resulting in unpredictable performance.

Hence, it is of critical importance to engineer predictable and theoretically sound low-level resource scheduling mechanisms that can provide stable and guaranteed performance to the various servers, even when they are executed on shared CPUs/cores. This result can be achieved by using real-time scheduling in the OS kernel or hypervisor, allowing for the temporal isolation among co-located services [28].

This resource scheduling mechanism can easily be integrated with standard QoS control policies for cloud services: thanks to the improved performance stability, performing the control actions becomes easier, making it possible to apply well-known control theory techniques to design the controller. As a result, in the proposed approach it is possible to dynamically change the scheduling parameters [29], introducing an additional knob that can be used by an orchestrator to fine-tune the CPU allocation to individual containers (vertical scalability) and achieve its control goals.

Therefore, the following of this paper focuses on isolating the performance of individual co-located services (isolated through containers) within a cloud platform, with particular reference to CPU scheduling, thus CPU-intensive services[4], as illustrated in Figure 2. The meaning of the per-container scheduling parameters (Q_i, P_i) will be clarified just below.

[4]For data-intensive services, the technique can be enriched by integrating additional QoS control mechanisms at the networking, disk or I/O layers.

Within the presented scenario, the proposed approach is tailored on the NFV use-case, where a set of VNFs is deployed as containers hosting packet processing servers across many possibly heterogeneous computing nodes. Each node is characterized by various timing requirements, due to different classes of traffic.

In the following, the proposed modifications to the Linux real-time scheduler are described and shown with a detailed modeling example. This allows for leveraging the gained predictability of the computing performance of single containers, as a function of the adopted scheduling parameters, to build a probabilistic performance model of a simple synthetic packet processing service. This way, statistics of the response-time distribution can easily be tied to the configuration of the server. Hence, it is possible to choose the server configuration parameters as a function of the desired QoS, allowing a resources controller to decide on the feasibility of a requested throughput based on the underlying resources and the already allocated services.

A. CPU scheduler for containers

Linux containers, that can be created by using a variety of user-space tools such as Docker, `lxc`, or similar, are associated with *control group*s (`cgroup`) and *namespace*s. In particular, a control group allows one to specify *limits* on the amount of resources (memory, CPU time, etc...) that the container can use. An important feature in this regard is the possibility to control the amount of time real-time tasks (`SCHED_FIFO` and `SCHED_RR`) running in the container can be scheduled for. Our modifications to the Linux scheduler allow for building theoretically-sound scheduling hierarchies through `cgroups`[5].

The Linux kernel provides the `SCHED_DEADLINE` CPU scheduling class [7], [30], implementing the Constant Band-

[5]The Linux kernel already provides hierarchical scheduling for real-time tasks, but its design aims only at acting as a limitation, not as a guarantee.

width Server (CBS) [31] algorithm, that allows to reserve a well defined amount of CPU time to a task (process or thread). In particular, the task is reserved an amount of time Q (`runtime`) every `period` P. The "Hierarchical CBS" (HCBS) extension implements hierarchical scheduling [23], [32] based on `SCHED_DEADLINE`. In particular, a CPU real-time reservation (Q, P) can be assigned to a control group as a whole (instead of a single task), to control the amount of CPU time reserved to real-time tasks running in the group. As a result, the `SCHED_DEADLINE` policy is used to select a control group to be scheduled on each CPU, and the `SCHED_FIFO` or `SCHED_RR` policy is used to select the tasks in the scheduled control group. The obtained mechanism, conceptually similar to [33] and [26], supports partitioned scheduling in the host (each `SCHED_DEADLINE` entity used to schedule a control group is bound to a CPU/core) and generic affinities in the guest (fixed priority tasks in the control group can have generic affinities; hence supporting both partitioned and global scheduling).

B. Probabilistic model

Consider a physical host with m identical pCPUs employing partitioned scheduling, where each pCPU hosts up to n containerized servers. Each server $i \in \{1, \dots, n\}$ receives a pattern of requests modeled as a Poisson stochastic process. Namely, requests are sent by the client with exponential and i.i.d. inter-request times with average rate λ_i, with requests of size s_i, and exponential and i.i.d. packet processing times with average rate μ_i. The latter is the processing time when the server is running on a whole CPU at maximum speed[6], but as explained earlier each server is assumed to be hosted in a container under reservation-based scheduling as explained above, using parameters Q_i and P_i.

The overall end-to-end round-trip time (RTT) for requests is thus a stochastic variable $R_i^e = t_i^S + t_i^P + t_i^R$, where:

1) t_i^S is the time to send the request from the client to server, including possible queuing time if there are pending earlier requests;
2) t_i^P is the time to process the request within the server, including queuing time if there are earlier requests;
3) t_i^R is the time to send the response back to the client, including queuing time if there are other requests waiting to be transmitted back.

The transmission and response times can be detailed as

$$t_i^S = q_i^S + \delta_i + \frac{z_i^S}{\sigma_i},\ t_i^R = q_i^R + \delta_i + \frac{z_i^R}{\sigma_i} \tag{1}$$

where q_i^S (q_i^R) is the queuing time during which a request is waiting to be transmitted (sent back), plus the client-server transmission latency δ_i (measurable as, e.g., half the ping time between the client and the server), plus the time needed to transmit a request (reply) of size z_i^S (z_i^R) on a medium with speed σ_i, assumed to be equal to a fraction of the available NIC speed (e.g., 1 Gbps or 10 Gbps), as specified in symmetric traffic/QoS control rules for incoming and outgoing packets for container i.

The SCHED_DEADLINE scheduler provides the guarantee [34] that, given a processing activity having a duration of C when running in isolation on a CPU, under a reservation (Q, P) the computation is finished within a time: $t^P = \lceil \frac{C}{Q} \rceil \cdot P$, with $\lceil \cdot \rceil$ denoting the ceiling function. Given a Q/P ratio, if the reservation period P is sufficiently small, the activity finishing time is reasonably approximated as:

$$t^P \cong \frac{C}{Q/P}, \tag{2}$$

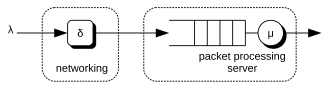
namely the activity behaves as if deployed on a virtual CPU that is slower of a factor of Q/P w.r.t. the pCPU it is deployed onto (Eq. (2) is valid with no queuing of activities).

When a single reservation is used to serve a second level scheduler (as in our proposed HCBS scheduler), the analysis is more complex [23], [35] but a "fluid approximation" similar to Eq. (2) can still be used. Considering a task set Γ scheduled by a reservation (Q, P), if a task $\tau \in \Gamma$ has response time R when running on a physical CPU together with the other tasks from Γ and P is sufficiently small, then the response time of τ can be approximated as

$$t^P \cong \frac{R}{Q/P}. \tag{3}$$

In practice, the period P can be set to as little values as a few tens of ms, but amounts smaller than $1ms$ would cause excessive scheduling overheads (and context switches).

Let us focus on the simplistic assumption of *uncongested network* first, where the client-server transmission latency δ_i has very little variability (a commonplace situation if the client resides within the same or a closeby data center, e.g., easily verified in NFV scenarios), requests are sufficiently inter-spaced so that the queuing times when sending requests and replies are negligible $q_i^S \cong q_i^S \cong 0$, and both requests and responses have a bounded size $z_i^S \le Z_i$ and $z_i^R \le Z_i$. Then, from Eq. (1) we have $t_i^S \cong \delta_i + \frac{Z_i}{\sigma_i}$, $t_i^R \cong \delta_i$, and we can substantially ignore the transmission and response times in the model, adding back the constant $2\delta_i + \frac{Z_i}{\sigma_i}$ in the final expression of the RTT. In what follows, for a networking symbol t, the following notation is used: $\tilde{t} \triangleq t - \delta$.



Under Poissonian arrivals with average rate λ_i and service times approximated as exponentially distributed with average rate $\mu_i \frac{Q_i}{P_i}$ (due to Eq. (2)), we have an approximate M/M/1 model. So, under the stability assumption of

[6]For the purposes of this paper, all the CPUs are assumed to be locked at their maximum speed. Power management and CPU frequency switching via DVFS are left out for the sake of simplicity, but they will be tackled in future works.

$\rho_i \triangleq \frac{\lambda_i}{\mu_i \frac{Q_i}{P_i}} < 1$, the following well-known results apply for t^P and its Cumulative Distribution Function (CDF) $F_{t^P}(\cdot)$:

$$E[t^P] = \frac{1}{\mu_i \frac{Q_i}{P_i} - \lambda_i}, \; F_{t^P}(t) = 1 - e^{-(\mu_i \frac{Q_i}{P_i} - \lambda_i)t}. \tag{4}$$

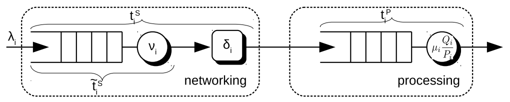
For example, given an input rate λ_i, the condition on the scheduling parameters (Q_i, P_i) for keeping a ϕ_i^{th} percentile of the response-time distribution R_i^e below a threshold D_i, to be intersected with the stability condition, is:

$$\begin{aligned} \Pr(R_i^e \le D_i) = 1 - e^{-(\mu_i \frac{Q_i}{P_i} - \lambda_i)(D_i - 2\delta_i)} \ge \phi_i \\ \frac{Q_i}{P_i} \ge \frac{1}{\mu_i}\left(\lambda_i - \frac{\ln(1-\phi_i)}{D_i - 2\delta_i}\right). \end{aligned} \tag{5}$$

Similarly, we can compute the maximum sustainable input rate λ_i with the specified probabilistic QoS constraint given an underlying pCPU with processing capability μ_i, or the minimum achievable deadline D_i with given λ_i, μ_i, and ϕ_i, as a function of (Q_i, P_i):

$$\lambda_i \le \mu_i \frac{Q_i}{P_i} + \frac{\ln(1-\phi_i)}{D_i - 2\delta_i}, \; D_i \ge 2\delta_i - \frac{\ln(1-\phi_i)}{\mu_i \frac{Q_i}{P_i} - \lambda_i}. \tag{6}$$

Assume now an exponential distribution of the request sizes s_i^S thus of the transmission times t_i^S with an average transmission rate $\nu_i = \frac{E[s_i]}{\sigma_i}$, and response-time approximated as $t_i^R \cong \delta_i$, as due to, e.g., sending back to the client just a success/error code (symmetrically, we can think of a $t_i^S \cong \delta_i$ and exponentially distributed s_i^R and t_i^R, as due to e.g., replying with data to a very short request packet). With exponentially distributed service times and under scheduling parameters (Q_i, P_i) Eq. (2) implies an average service rate of $\mu_i \frac{Q_i}{P_i}$.



Then, the system can be approximated as a sequence of two M/M/1 queues, which, under the stability condition of:

$$\lambda_i < \nu_i \wedge \lambda_i < \mu_i \frac{Q_i}{P_i}, \tag{7}$$

has the steady-state behavior of two independent M/M/1 queues, with the process of arrivals at the server input queue being Poissonian with the same parameter λ_i. Therefore, denoting with $\tilde{t}_i^S$ the time needed for networking where the constant term δ_i has been removed, we have:

$$R_i^e = 2\delta_i + \tilde{t}_i^S + t_i^P, \qquad E[\tilde{t}_i^S] = \frac{1}{\nu_i - \lambda_i}, \tag{8}$$

$$E[R_i^e] = 2\delta_i + \frac{1}{\nu_i - \lambda_i} + \frac{1}{\mu_i \frac{Q_i}{P_i} - \lambda_i}, \qquad E[t_i^P] = \frac{1}{\mu_i \frac{Q_i}{P_i} - \lambda_i}. \tag{9}$$

Imposing the probabilistic QoS constraint on R_i^e, as due to the sum of two exponential distributions with different parameters, we can easily get to a closed-form bound of R_i^e splitting it proportionally to the expected values of the two components. Formally, using the easy-to-prove lower-bound

$$\Pr[X + Y \le z] \ge \Pr\left[X \le z \frac{E[X]}{E[X] + E[Y]} \wedge Y \le z \frac{E[Y]}{E[X] + E[Y]}\right],$$

we have:

$$\Pr[R_i^e \le D_i] = \Pr[\tilde{t}_i^S + t_i^P \le D_i - 2\delta_i] \ge \left(1 - e^{-\alpha_i(D_i - 2\delta_i)}\right)^2 \tag{10}$$

where $\alpha_i \triangleq (\frac{1}{\nu_i - \lambda_i} + \frac{1}{\mu_i \frac{Q_i}{P_i} - \lambda_i})^{-1}$, and $\tilde{t}_i^S$ and t^P have been assumed to be i.i.d. and independent from one another. Therefore, we can compute, e.g., the minimum deadline D_i guaranteeing $\Pr[R_i^e \le D_i] \ge \phi_i$:

$$\begin{aligned} \Pr[R_i^e \le D_i] \ge \left(1 - e^{-\alpha_i(D_i - 2\delta_i)}\right)^2 \ge \phi_i \\ D_i \ge 2\delta_i - \frac{\ln\left(1 - \sqrt{\phi_i}\right)}{\alpha_i}. \end{aligned} \tag{11}$$

Note that $\nu_i \to \infty$ leads to an expression similar to Eq. (6), just with $\sqrt{\phi_i}$ rather than ϕ_i, providing an insight into the implications of the approximated bound of Eq. (10) above. The minimum α_i satisfying the probabilistic constraint is: $\alpha_i \ge -\frac{\ln\left(1 - \sqrt{\phi_i}\right)}{D_i - 2\delta_i}$, leading to the maximum input rate λ_i sustainable with the given configuration (computed as a solution of a 2nd order inequality):

$$\begin{cases} \lambda_i & \le \frac{\mu_i \frac{Q_i}{P_i} + \nu_i}{2} + \beta_i \left[\sqrt{1 + \left(\frac{\mu \frac{Q_i}{P_i} - \nu_i}{2\beta_i}\right)^2} - 1\right] \\ \beta_i & = -\frac{\ln\left(1 - \sqrt{\phi_i}\right)}{D_i - 2\delta_i}. \end{cases} \tag{12}$$

IV. Experimental results

This section presents results validating the approach presented in Section III, by means of experimental validation, carried out using a Linux kernel v4.16.0-rc1, modified with our HCBS patch described in Section III-A, and Linux containers through `lxc`, where we have set per-container HCBS parameters (Q, P) as needed for each experiment. Moreover, the proposed reservation approach has been tested using a simple yet realistic NFV application.

A. Validation of the model

In the first set of experiments, the stochastic model presented in Section III-B has been validated using an open-source distributed application we wrote, `distwalk`[7], able to impose a configurable client-server networking traffic and processing workload on the server. When imposing a specific average CPU workload LD, under an average request arrival rate of λ, we have configured the server to keep an average per-request processing time equal to $1/\mu = LD/\lambda$.

[7]More information is available at: https://github.com/tomcucinotta/distwalk.

In what follows, every measurement has been obtained from 10K samples over each run, with the affected machines running with CPU frequency switching and hyper-threading disabled. The machine used for the server was an Intel(R) Core(TM) i7-4790K CPU @ 4.00GHz with 16GB of RAM. Clients have been run on a laptop equipped with an Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz and 16GB of RAM. Machines had a $1Gbps$ NIC connected to an L2 switch. The client-server latency has been measured using `ping` and gathering 10K samples, obtaining $\delta = 252/2 = 126\,\mu s$.

Negligible sending time model: Figure 3 shows results from a set of experiments that have been run, with various values of the computational workload (from 0.5 to 0.8, one for each plot), the average request inter-arrival time (from $800\mu s$ to $5ms$, on the X-axis), and a small packet size of $S = 128$ bytes making networking time negligible w.r.t. processing at $\sigma = 1Gbps$ ($S/\sigma \cong 1\mu s$). Note that, as in each plot the computational workload is kept constant (denoted with LD on top), increasing the average inter-arrival times (X-axis) implies a corresponding increase of the average processing times on the server, thus of the response times, as evident from the plots.

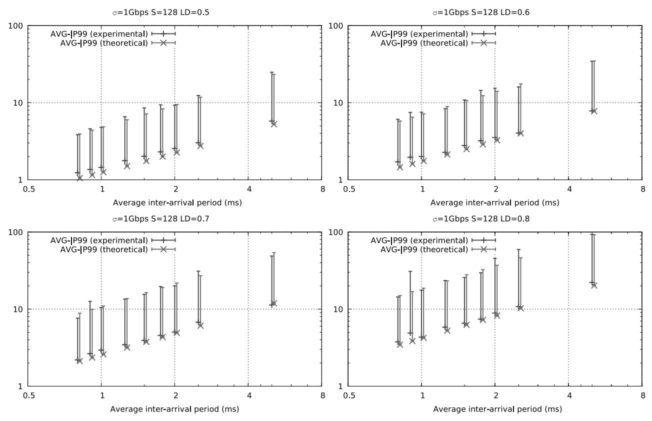


Figure 3. Experimental response times (in ms) for various configurations, compared with the theoretical expectations (negligible sending time case). Segments connect average to the 99^{th} percentile of response-time distributions.

The average and 99^{th} percentile of the experimental response-times are reported in the plots, along with the theoretical expectations as from Eq. (4) and Eq. (6), for the average and 99^{th} percentile, respectively. As visible, theory matches quite closely with theoretical expectations.

Impact of scheduling parameters (Q, P): Figures 4 (a)..(d) show how the values of the scheduling parameters (Q, P) allow for controlling the expected response-time statistics. The set of tried configurations have negligible networking time, with S ranging from $64B$ to $1024B$ (with $1/\lambda = 2ms$ in (c) this is an average throughput of 4 Mbps over a $1Gbps$ network).

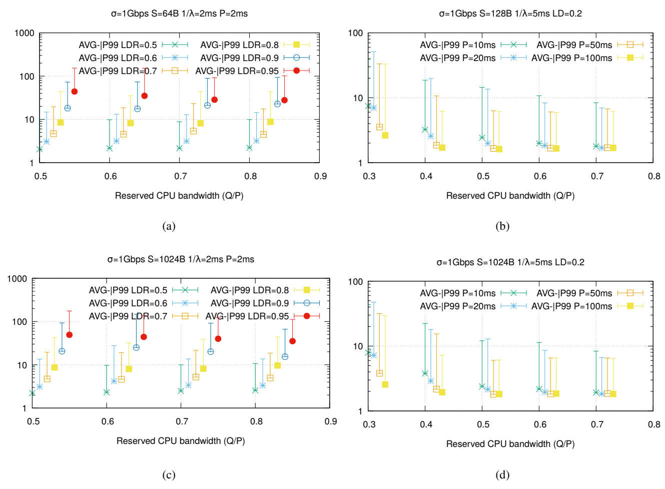


Figure 4. Experimental response-time statistics (on the Y-axes, in ms), where vertical segments connect the average and 99^{th} percentile for each configuration: with fixed reservation period but varying reserved CPU bandwidth (on the X-axis) and workload ratio (different curves) in plots (a) and (c) on the left; with fixed CPU load but varying reserved CPU bandwidth (X-axis) and reservation period (different curves) in plots (b) and (d) on the right.

In Figures 4 (a) and (c), obtained with an arrival period of $1/\lambda = 2ms$ and reservation period fixed at $P = 2ms$, we used a Q/P reserved bandwidth going from 0.5 to 0.9 (on the X axis), serving a computational workload filling up the reservation from a load ratio (LDR) from 0.5 to 0.95 (multiple curves in correspondence of each Q/P value, slightly inter-spaced horizontally to enhance readability). Note that the load ratio LDR is related to the computational workload LD and reservation bandwidth (Q, P) by: $LDR = LD/(Q/P)$, and that $LDR = 1$ would be the meta-stability region. The curves show that the response-time grows as the load ratio increases, while it stays basically flat at constant load ratio. Figures 4 (b) and (d) show results with an average inter-arrival time of $1/\lambda = 5ms$, and an average computational workload of $LD = 20\%$, at varying values of the reserved CPU bandwidth Q/P, which needs to be greater than 0.2, for the system to be stable. Differently from Figures (a) and (c) where curves are obtained at constant load ratio (so changing Q/P implies changing μ as well), in (b) and (d) μ is kept constant, thus increasing Q/P the experimental response-times decrease, as expected. The figures also report, for each Q/P value, results with different P values (multiple closeby segments in correspondence of each Q/P value, slightly spaced horizontally to enhance readability). Better results are obtained for higher P values. Decreasing P does not make a visible difference at a CPU reservation sufficiently higher than the minimum needed for stability, while it remarkably worsens the response times with the smaller reservations of $Q/P = 40\%$ and 30%. This happens because of the additional scheduling overheads (being 10 times bigger for $P = 10ms$ than they are with $P = 100ms$), that become non-negligible at low reserved utilizations (and runtimes, as $P = 10ms$ and $U = 0.3$ imply a runtime of $Q = 300\mu s$ and 100 context switches per second). Overall, these experiments confirm computing predictability under given reservation parameters.

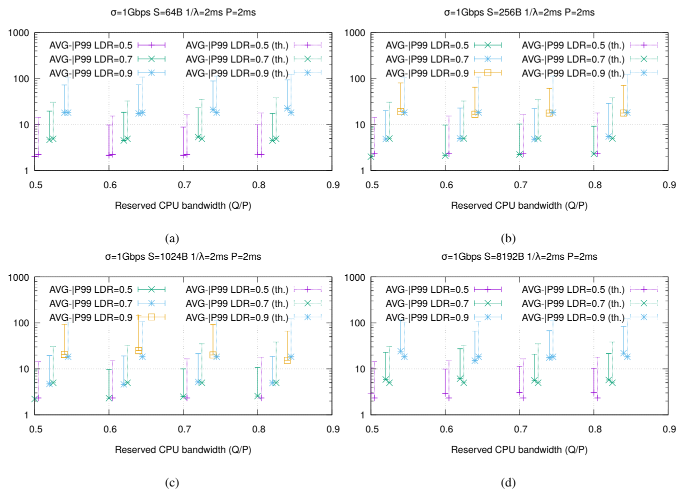


Figure 5. Experimental response-time statistics (on the Y-axes, in ms) versus theoretical expectations, where vertical segments connect the average and 99th percentile for each configuration, with different plots representing results for different packet sizes, at varying reserved CPU bandwidth (X-axis) and for various computational workload ratio (LDR) (different curves).

Non-negligible sending time model: Figures 5 (a)..(d) show results compared with the theoretical average and 99th percentile estimates obtained through the models of Eq. (9) and Eq. (11), in the cases of very small packets ($S = 64B$ in (a) and $S = 256B$ in (b)), larger packets still fitting in an Ethernet frame ($S = 1024$ in (c)), or significantly larger packets ($S = 8192$ in (d)). As evident from the plots, the predicted average value is quite tight, while the predicted 99th percentile is somewhat over-estimated in this case, as due to the bounded computations in Eq. (11). However, the estimate is conservative, and the actual experimental values are safely below the prediction, which gives some robustness w.r.t. possible unmodeled effects.

IMS test case: A possible NFV application that benefits from the usage of the described reservation approach is a SIP connections manager, widely used for VoIP applications. The following example compares the experimental cumulative distribution functions (ECDF) of the RTTs obtained by using Kamailio[8] to manage SIP traffic generated by the SIPp[9] tool.

To prevent interference with other running tasks, the client runs SIPp as a high priority, `SCHED_FIFO` task, and the server runs in a 1-vCPU container hosting the Kamailio and Mysql processes. SIPp is configured to perform 257 registrations every $100ms$, reporting the individual measured RTT times. For each registration request, the used scenario involves two packet round-trips and the interaction with the Mysql database.

Fig. 6 shows the results achieved in the different cases: the *CFS* plot corresponds to the use of the default Linux CFS scheduler, which, splitting the CPU time equally among the active tasks, allows Kamailio and Mysql to take the whole time of the underlying pCPU. The *CFS noise* plot shows the results achieved when in the system coexist four periodic rt-app[10] tasks running for $6ms$ every $100ms$. Since a portion of the CPU time is assigned to them, the performance is strongly affected, resulting in less than 70% of SIP RTTs measured before $2ms$.

The *HCBS* plot shows the RTTs when Kamailio and Mysql are run as `SCHED_RT` tasks within a HCBS container, tuned with a runtime of $1.4ms$ and period of $2ms$, which satisfies the average load (about 60%) with some safety margin. For the *HCBS noise* case, the same rt-app disturbance has been used as for the *CFS noise* case, this time running as `SCHED_RR`, within another HCBS server, with runtime $24ms$ and period $100ms$. Since `SCHED_DEADLINE` is the highest priority scheduling class, *HCBS* average performance is slightly better than *CFS*, but, due to the throttling when the group budget is exhausted, the 99th percentile of *HCBS* is at $1287\mu s$, while for the *CFS* is at $980\mu s$. Due to the reservation mechanism, the rt-app tasks only have a marginal influence on the ECDF, apart from the overhead introduced by the context switches. In the presence of interfering tasks *HCBS noise* has a 99th percentile at $1268\mu s$ (higher than *HCBS* because of measurement noise), *outperforming CFS* that reaches that percentile *one order of magnitude later*, at $10614\mu s$.

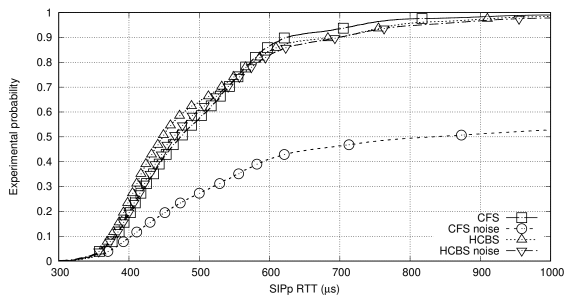


Figure 6. ECDF plots of SIPp authentication RTT using the CFS algorithm and HCBS, with and without interfering tasks.

V. Conclusions

This paper introduced a new solution for implementing distributed Cloud services characterized by stringent timing

[8]More information at: http://kamailio.org/.

[9]More information at: http://sipp.sourceforge.net/.

[10]More information at: https://github.com/scheduler-tools/rt-app/.

and resource constraints like those deployed at the edge of the network (e.g., NFV placement in VRAN). As an example, an implementation based on Linux containers has been presented, but other implementations are also possible. The proposed solution leverages existing real-time theory to achieve predictable QoS, useful in time-critical applications (e.g., 5G network function split), making it possible to provide a simple analysis based on queueing theory.

As future work, a feedback mechanism will be added to adapt the resource allocation to dynamic workloads and to cope with overload conditions. Moreover, the theoretical analysis will be extended to consider more complex scenarios. Inter-container communications are planned to be optimized and applicability to RAN software components such as OpenAirInterface is among the planned investigations.

References

[1] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft, "Unikernels: Library operating systems for the cloud," *SIGARCH Comput. Archit. News*, vol. 41, no. 1, pp. 461–472, Mar. 2013.

[2] A. Kantee, "The rise and fall of the operating system," *USENIX login*, vol. 40, no. 5, October 2015.

[3] G. Banga, P. Druschel, and J. C. Mogul, "Resource containers: A new facility for resource management in server systems," in *OSDI*, vol. 99, 1999, pp. 45–58.

[4] H. Kang, M. Le, and S. Tao, "Container and microservice driven design for cloud infrastructure devops," in *2016 IEEE International Conference on Cloud Engineering (IC2E)*, April 2016, pp. 202–211.

[5] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski, and P. Suter, *Serverless Computing: Current Trends and Open Problems*. Singapore: Springer Singapore, 2017, pp. 1–20.

[6] J. Bhimani, Z. Yang, M. Leeser, and N. Mi, "Accelerating big data applications using lightweight virtualization framework on enterprise cloud," in *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, Sept 2017, pp. 1–7.

[7] J. Lelli, D. Faggioli, T. Cucinotta, and G. Lipari, "An experimental comparison of different real-time schedulers on multicore systems," *J. Syst. Softw.*, vol. 85, no. 10, pp. 2405–2416, Oct. 2012.

[8] A. Ali-Eldin, J. Tordsson, and E. Elmroth, "An adaptive hybrid elasticity controller for cloud infrastructures," in *2012 IEEE Network Operations and Management Symposium*, April 2012, pp. 204–212.

[9] N. Roy, A. Dubey, and A. Gokhale, "Efficient autoscaling in the cloud using predictive models for workload forecasting," in *2011 IEEE 4th International Conference on Cloud Computing*, July 2011, pp. 500–507.

[10] K. Konstanteli, T. Cucinotta, K. Psychas, and T. Varvarigou, "Admission Control for Elastic Cloud Services," in *2012 IEEE Fifth International Conference on Cloud Computing*, June 2012, pp. 41–48.

[11] K. Konstanteli, T. Cucinotta, K. Psychas, and T. A. Varvarigou, "Elastic admission control for federated cloud services," *IEEE Transactions on Cloud Computing*, vol. 2, no. 3, pp. 348–361, July 2014.

[12] M. Alicherry and T. V. Lakshman, "Network aware resource allocation in distributed clouds," in *2012 Proceedings IEEE INFOCOM*, March 2012, pp. 963–971.

[13] T. Cucinotta, D. Lugones, D. Cherubini, and E. Jul, "Data Centre Optimisation Enhanced by Software Defined Networking," in *2014 IEEE 7th International Conference on Cloud Computing*, June 2014, pp. 136–143.

[14] T. Cucinotta, F. Checconi, Z. Zlatev, J. Papay, M. Boniface, G. Kousiouris, D. Kyriazis, T. Varvarigou, S. Berger, D. Lamp, A. Mazzetti, T. Voith, and M. Stein, "Virtualised e-Learning with real-time guarantees on the IRMOS platform," in *2010 IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, Dec 2010, pp. 1–8.

[15] J. Lee, S. Xi, S. Chen, L. T. X. Phan, C. Gill, I. Lee, C. Lu, and O. Sokolsky, "Realizing compositional scheduling through virtualization," in *2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium*, April 2012, pp. 13–22.

[16] X. Yi, J. Duan, and C. Wu, "GPUNFV: A GPU-Accelerated NFV System," in *Proceedings of the First Asia-Pacific Workshop on Networking*, ser. APNet'17. New York, NY, USA: ACM, 2017, pp. 85–91.

[17] X. Ge, Y. Liu, D. H. Du, L. Zhang, H. Guan, J. Chen, Y. Zhao, and X. Hu, "OpenANFV: Accelerating Network Function Virtualization with a Consolidated Framework in Openstack," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14. New York, NY, USA: ACM, 2014, pp. 353–354.

[18] T. Cucinotta, L. A. M. Marinoni, and C. Vitucci, "The Importance of Being OS-aware - In Performance Aspects of Cloud Computing Research," in *Proceedings of the 8th International Conference on Cloud Computing and Services Science*, 2018.

[19] C. Vitucci and A. Larsson, "Flexible 5G Edge Server for Multi Industry Service Network," *International Journal on Advances in Networks and Services*, vol. 10, no. 3-4, 2017, ISSN: 1942-2644.

[20] X. Costa-Perez, J. Swetina, T. Guo, R. Mahindra, and S. Rangarajan, "Radio access network virtualization for future mobile carrier networks," *IEEE Comm. Magazine*, vol. 51, no. 7, pp. 27–35, July 2013.

[21] B. Haberland, F. Derakhshan, H. Grob-Lipski, R. Klotsche, W. Rehm, P. Schefczik, and M. Soellner, "Radio base stations in the cloud," *Bell Labs Technical Journal*, vol. 18, no. 1, pp. 129–152, June 2013.

[22] "3rd Generation Partnership Project; Transport requirement for CU-DU functional splits options; R3-161813 (document for discussion)," in *3GPP TSG RAN WG3 Meeting 93*, August 2016.

[23] I. Shin and I. Lee, "Compositional real-time scheduling framework," in *25th IEEE International Real-Time Systems Symposium*, Dec 2004, pp. 57–67.

[24] S. Xi, C. Li, C. Lu, C. D. Gill, M. Xu, L. T. X. Phan, I. Lee, and O. Sokolsky, "RT-Open Stack: CPU Resource Management for Real-Time Cloud Computing," in *2015 IEEE 8th International Conference on Cloud Computing*, June 2015, pp. 179–186.

[25] Y. Li, L. T. X. Phan, and B. T. Loo, "Network functions virtualization with soft real-time guarantees," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, April 2016, pp. 1–9.

[26] A. Parri, M. Marinoni, J. Lelli, and G. Lipari, "An implementation of a multiprocessor bandwidth reservation mechanism for groups of tasks," in *Proceedings of the 16th Real Time Linux Workshop*, OSADL, Ed., Dusseldorf, Germany, October 2014.

[27] L. Abeni, A. Balsini, and T. Cucinotta, "Container-based real-time scheduling in the linux kernel," in *Proceedings of the Embedded Operating System Workshop 2018 (EWiLi'18)*, Torino, Italy, October 2018.

[28] T. Cucinotta, L. Abeni, M. Marinoni, A. Balsini, and C. Vitucci, "Virtual network functions as real-time containers in private clouds," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, July 2018, pp. 916–919.

[29] L. Abeni, T. Cucinotta, G. Lipari, L. Marzario, and L. Palopoli, "Qos management through adaptive reservations," *Real-Time Systems*, vol. 29, no. 2, pp. 131–155, Mar 2005. [Online]. Available: https://doi.org/10.1007/s11241-005-6882-0

[30] J. Lelli, C. Scordino, L. Abeni, and D. Faggioli, "Deadline scheduling in the Linux kernel," *Software: Practice and Experience*, vol. 46, no. 6, pp. 821–839, 2016, spe.2335.

[31] L. Abeni and G. Buttazzo, "Integrating multimedia applications in hard real-time systems," in *Proceedings of the IEEE Real-Time Systems Symposium*, Madrid, Spain, December 1998.

[32] G. Lipari and E. Bini, "A framework for hierarchical scheduling on multiprocessors: from application requirements to run-time allocation," in *Proc. of 31st IEEE Real-Time Systems Symposium*, Dec. 2010, pp. 249–258.

[33] F. Checconi, T. Cucinotta, D. Faggioli, and G. Lipari, "Hierarchical Multiprocessor CPU Reservations for the Linux Kernel," in *Proceedings of the 5th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT 2009)*, June 2009.

[34] L. Abeni and G. Buttazzo, "Stochastic analysis of a reservation based system," in *Proc. of the 9th International Workshop on Parallel and Distributed Real-Time Systems*, vol. 1, 2001.

[35] S. Saewong, R. R. Rajkumar, J. P. Lehoczky, and M. H. Klein, "Analysis of hierarchical fixed-priority scheduling," in *Proceedings of the 14th Euromicro Conference on Real-Time Systems*, ser. ECRTS '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 173–.